



User Guide of Cargo Data Miner

Xuzhou Qin

10 August 2016

Contents

1	Introduction	4
2	Prerequisites	4
2.1	Directory Structure	5
2.2	Installation on a PC	6
2.3	Deployment on a Shiny Server Pro	7
3	Structure and Features	7
3.1	Cargo IS Database	8
3.1.1	Data loader	8
3.1.2	Top airports/air routes	10
3.1.3	Treemap	12
3.1.4	Yield Evolution	13
3.1.5	Air Waybill Analysis	15
3.1.6	Yield Calculator	17
3.2	Seabury Database	17
3.2.1	Data loader	18
3.2.2	Commodity treemap	19
3.2.3	Evolution plot	20
3.3	FlightRadar 24 Database	20
3.3.1	Data loader	21
3.3.2	Airline network	21
3.4	Other features	21
3.4.1	Connection status check	21
3.4.2	Fast selectize input	22

4	Future Development	22
5	Appendix: Troubleshooting	22
5.1	Install Oracle Database instance client	22
5.2	Download and install an R package manually	23

Abstract

This is the user guide of *Cargo Data Miner*. *Cargo Data Miner* is an R-Shiny Dashboard application developed by Airbus Freighter Marketing department (CSMX). The initiative is to build a platform where everybody could query and analyze cargo data. In this document, detailed information and instruction are provided in order to facilitate the usage and maintenance of the application. For more information, the author could be reached at xuzhou.qin@gmail.com.

Acknowledgement

The present work benefited from Oliver Von-Tronchin, tutor of my internship and director of Freighter Marketing (CSMX), Sebastian Aponte, Nicolas Chauviere-Courcol, colleagues of mine in CSMX, Federico Semeraro and Andy Williams, for their inspiration of the project *LiMA*, Pierre Binet, Didier Gauzy, Aurelien Turina, Sam Crawshay Jones, Lionel Cousseins, Yves Renard and all my colleagues of Airbus who provided valuable comments and ideas to the undertaking of the project here.

1 Introduction

Aviation market is a highly complex and data-dependent market. Inside Airbus, numbers of different servers are deployed to stock, manage and provide database services to support our daily work. It is striving to introduce data digitalization as a core task of its future plan. The raising importance of data has already grabbed the attention of the managers and it will never stop growing. However, face to the massive amount of data, challenge still remains.

At the company level, the main challenge comes from the database management. Too many databases are stocked separately with different **database management system** (DBMS). The databases are quite isolated from the others. In other words, except people who frequently use one database, others could not even know its existence. Then, the different DBMS cause inconvenience in database management. For example in the freighter marketing department (CSMX), three marketing databases, *Cargo IS*, *Seabury* and *FlightRadar 24* are stocked separately in three data servers using three different DBMS (Oracle Database, MySQL and Microsoft SQL server).

At the department level, the major challenge is the lack of an efficient and user-friendly tool for the data extraction and analysis process. Even though the traditional approach of database management, SQL, is widely used for data manipulation. But in order to use it efficiently, you must have some knowledge of programming and you have to suffer the difficulty in interfacing. As SQL is not a common tool inside marketing department, interacting with the data by using SQL may not be a suitable choice. As to the data analysis tool, Excel is one of the most widely used software. But it suffers from some innate limitations facing the increasing amount of data.

There are also some Airbus-made tools for marketing department, like *Acper*, *OAG Builder*, *route06*, etc. These are very powerful and sophisticated programs dedicated for specific usage since a very long time. They are irreplaceable for some tasks but they are still not perfect. For example, in terms of learning difficulty, they both have a steep learning curve. Some special training session are needed to master these tools. And from a graphical design's point of view, they often remind me about the "old fashioned" software interface, which may be not enough intuitive and user-friendly. The need to visualize the analytic results in a modern way is thus urgent.

Due to these facts, data digitalization inside Airbus should be done from two levels. We will need an uniform and standardized platform to stock and manage data at the company level. And inside the departments, new data mining and analysis tools are needed to support the works in the big data era. Some moves have already been realised inside Airbus. **BIO** is a platform which provides an automatic aggregation of different databases. **Foundry** is a cloud-based platform for collaboratively big data analysis built by Palantir Technologies.

This document provides a solution for the development of a data analysis tool by using **R Shiny Dashboard**. **Shiny Dashboard** is a framework to create a light weight, powerful and interactive web application. It uses a reactive programming model so that outputs change instantly as users modify inputs. It is easy to use, no web designing language required. But further customization could be easily done with some HTML, CSS and JavaScript knowledge.

In the first section of this document, installation and deployment prerequisites are presented. Then, detailed functionalities and the application structure are stated in section two in order to provide technical support for using and maintenance. In the third section, some suggestions for the future development of this application are proposed. The technical appendix could be found in the last section of the document.

2 Prerequisites

Cargo Data Miner (CDM) was developed with **Microsoft R Open 3.2.5**, which is an enhanced distribution of R. It supports by default multi-thread computing and it is perfectly compatible with the original R. The user interface used during the development is **RStudio**, which is the most common IDE of R.

To make **CMD** work properly, there are two main methods.

- Install and run it locally on a PC for personal use.
- Deploy it on a Shiny server so that CDM could be reached by an IP adress.

In this section, I will present both the two methods.

2.1 Directory Structure

The following figure presents the structure of the CDM directory.

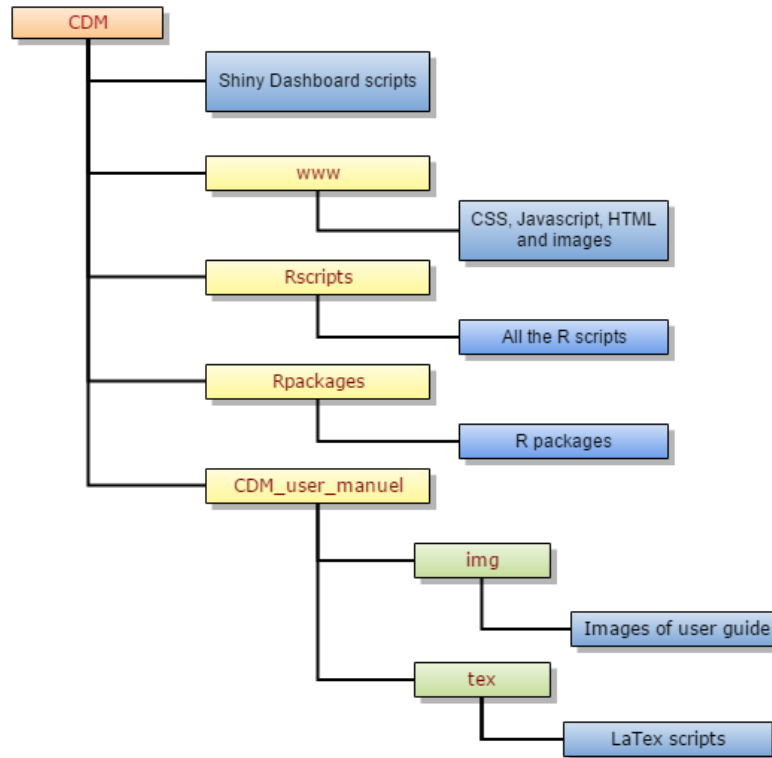


Figure 1: CDM directory structure

The main directory `~/CDM` contains Shiny Dashboard scripts (*ui.R*, *server.R* and *global.R*) and three directories:

- `/www`
- `/Rscripts`
- `/Rpackages`
- `/CDM_user_manuel`

All the images, HTML, CSS and JavaScript used in **CDM** should be placed in `~/CDM/www`. All the R scripts called by the application should be placed in `~/CDM/Rscripts` and all the R packages (for local installation) should be placed in `~/CDM/Rpackages`. These three directories are the essential parts of **CDM**.

In `~/CDM/CDM_user_manuel`, you will find the code of this use guide (written in R markdown) and its resources. All figures appeared in this document should be placed in `~/CDM/CDM_user_manuel/img` and all the LaTeX scripts called from the R markdown code will be found in `~/CDM/CDM_user_manuel/tex`.

2.2 Installation on a PC

CDM could run on a personal computer having installed R and a web browser with the correct configuration. Before the first use, some steps have to be followed.

- Step 1: Get R and other required software:
 - Install **Microsoft R Open 3.2.5** from <https://mran.revolutionanalytics.com/download/> (to turn on the multi-thread computing support, install the MLK package as well), or you can install the original **R** (64-bit) from <https://cran.r-project.org/mirrors.html>.
 - Install **RStudio** from <https://www.rstudio.com/products/rstudio/>. This is optional but highly recommended.
 - **CDM** works with Internet Explorer but you could also install Google Chrome for the best performance.
 - Install **Oracle Database Instant Client (11.2.0.3) EN_W764** from PC service. Or you can download it from Oracle website (Make sure it is the 64-bit version). If error of Oracle occurs during the execution of **CMD**, see Appendix: Troubleshooting for help.
- Step 2: Connection Setup
 - **CMD** will need the access right of three different databases. They are: **p595dodmp01** (Cargo IS, Seabury), **fr0-bio-p01** (BIO) and **fr0-dmds-p01** (FlightRadar 24). The user name and password are already pre-filled and there is no need to change, except for **BIO** database. Because you will need to grant the access right to your own *Windows* account to make the connection. To do so, you could ask *Aurelien Turina* for further information.
 - `~/CDM/global.R` contains the connection parameters of these databases.

```
#####
#' CONNECTION PARAMETERS
#'
#' Modify it before the first execution of the application
#####
#' ROracle connection string
#' for Cargo IS, Seabury
host = 'p595dodmp01'
port = 1521
sid = 'DBUPA269'
username_cargois <- 'CARGO'
password_cargois <- '*****'

#' RODBC MS SQL server connection string
#' for BIO
driver_bio <- 'SQL Server'
server_bio <- 'fr0-bio-p01'
port_bio <- 10335
username_bio <- ''
password_bio <- ''
trusted_connection_bio <- TRUE # if TRUE connect with windows login and pw

#' RMySQL MySQL connection string
#' for FlightRadar 24
```

```

user_fr24 <- "user_ext"
password_fr24 <- "*****"
dbname_fr24 <- 'FR24'
host_fr24 <- "fr0-dmds-p01"

```

- Step 3: Now you can open RStudio.
 - Before running **CDM**, if you have more than one version of R installed on your computer, make sure that the R version is [64-bit] ~/MRO_3.2.5 (the name could be varied)
 - Click on the **run** button on the top right of the text editor aera to execute the application. The application will automatically check if all the required R packages are installed and it will install those which are not installed from **CRAN** (Comprehensive R Archive Network). **CRAN** is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. However, due to the internet connection configuration inside Airbus, if any of the required packages fails to download, you will have to download and install it manually (See Appendix for detail).

Now you are ready to use **CDM** :)

2.3 Deployment on a Shiny Server Pro

Shiny Server Pro is a server program that makes Shiny application available over the web.

At this time, Shiny Server can be run on Linux servers with explicit support for Ubuntu 12.04 or greater (64 bit) and CentOS/RHEL 5 (64 bit) or greater. Multiple Shiny applications on multiple web pages could be hosted with the same Shiny Server.

It can be downloaded from <https://www.rstudio.com/products/shiny/shiny-server/>.

For the detail instruction of the configuration and installation of Shiny Server Pro, see **Shiny Server Professional v1.4.2 Administrator's Guide** on <http://docs.rstudio.com/shiny-server/>.

3 Structure and Features

A Shiny dashboard application could contains three sources scripts:

- a user-interface script: `ui.R`
- a server script: `server.R`
- a global environment script: `global.R`

There is another structure of Shiny application where all the three main scripts could be integrated into one single script `app.R`. But here we only talk about the “traditional” structure.

The UI script controls the layout and appearance of the app. The server script contains the instructions that your computer needs to build your app. And the glabal script defines objects in the global scope.

Reactive programming is the fundamental of a Shiny Dashboard application. The main idea is to make it easy to wire up *input values* from a web page, trigger the R code to be (re)executed, and then have the results of your R code be written as *output values* back to the web page.

The following figure shows the three elements in Shiny reactive programming. **Reactive source** typically is the user's input through a web browser. It could trigger the execution of an R code. The output of the code is represented by the **reactive endpoint**. **Reactive conductor** could be placed between the sources

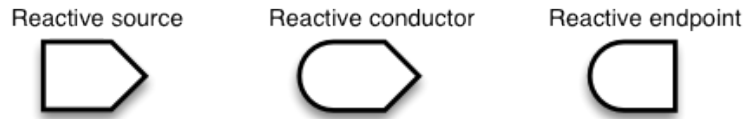


Figure 2: Essential objects in reactive programming

and endpoints. Generally they are used to encapsulate the computationally expensive operations in order to augment the code performance.

After launching **CDM**, you will see the following welcome page. **CDM** has three parts: a header, a sidebar and a body. The header show the application name and some notification¹, for example the database connection status, the app version and the developing progres. The sidebar could help you to navigate between different tabs The functionalities of these widgets will be presented in the following sections. Finally the body is the main aera where you will be able to interact with **CMD** and see the output.



Figure 3: CMD home page

Five different tabs are displayed on the sidebar. Each tab contains also several subtabs.

3.1 Cargo IS Database

Cargo IS database is the first major component of **CDM**. It provides airport level Air Waybill data, including airfreight charges, weight, numbers of shipmens and it is the only ressource of cargo yield information for Airbus.

3.1.1 Data loader

3.1.1.1 Instruction

¹Some features are still under development.

The first step is about how to load Cargo IS data. In the first dropdown menu, we could select the level of data granularity to be queried from Cargo IS (*e.g.*, airport to airport, country to country, region to region, etc). Then the year and the **O&D** (**O**rigin and **D**estination) of airfreight should be precised. You can check the box “Plot evolution since 2010” to load data of all the available year (2010-2016). When the box is checked, the year input will be ignored.

After selecting the right parameters, click on **Load Data** button. Some summerize of the queried dataset will be shown on the top of this tab. A preview of the 50 first rows will appear on the right. You could do a quick check of the data. The data table could be downloaded by simply clicking on the **Download Data** button.

Now the data table is loaded and ready to be used!

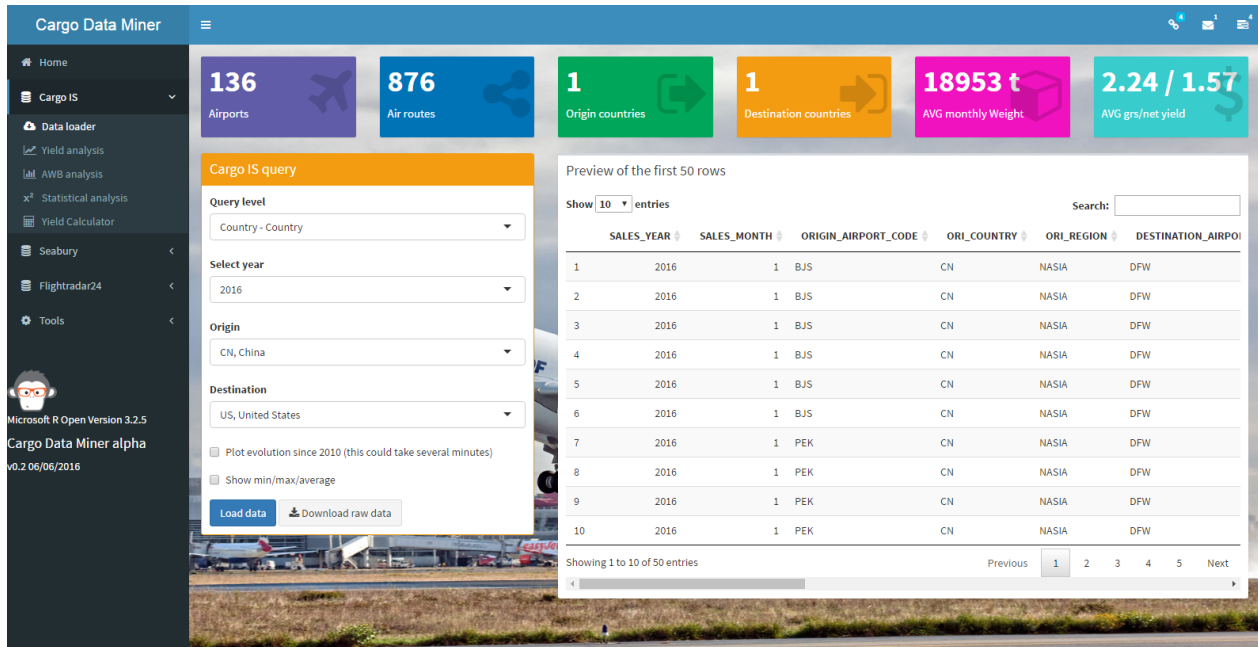


Figure 4: An example query of the 2015 airfreight from *China* to the *USA*.

3.1.1.2 Explanation

The data loading procedure will call the `CargoIS.SQL.query` function. The function will take the user parameters as input, generate an appropriate `Oracle SQL` code and send the query to the data server. The returned dataset is thus stocked in the global scope and it is ready to be used by other functions.

```
# EXAMPLE
# Query airfreight data of 2015 with origin country = China, destination = USA
dat <- CargoIS.SQL.query(CargoDB, level = 'C2C',
                        ORG = 'CN',
                        DST = 'US',
                        year = 2015)
```

The six value boxes on the top of the page are created by the function `valueBox(value, subtitle, icon, color, width, href)` based on the output dataset. They represent:

- **Airports**: Numbers of unique airports appeared in the dataset.
- **Air routes**: Numbers of unique airport pairs (Origin + Destination) in the dataset.

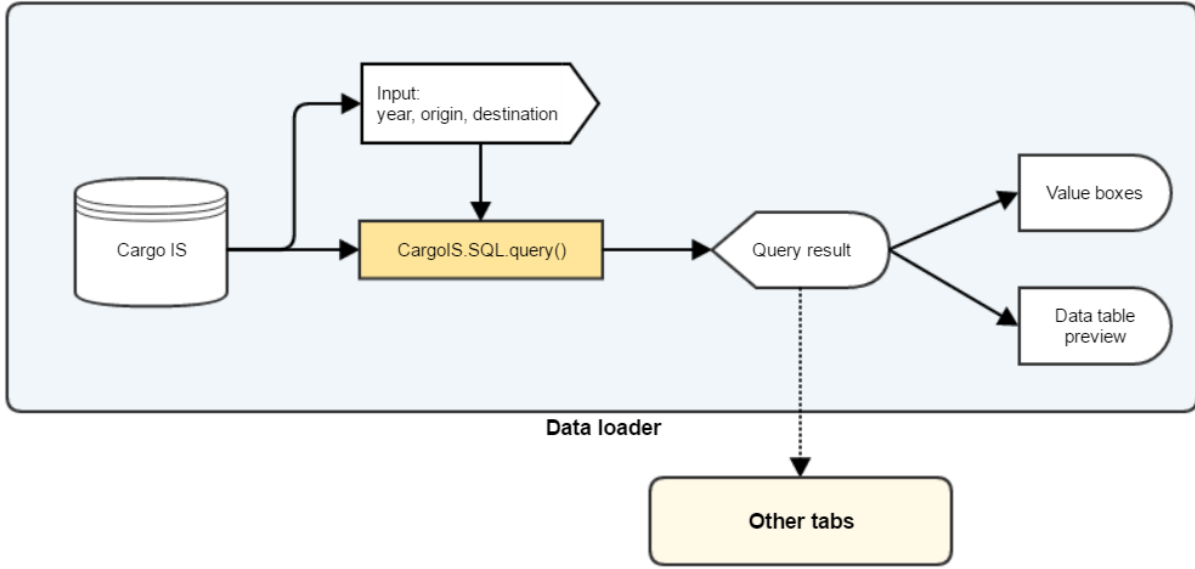


Figure 5: Flowchart of data loading process

- **Origin country:** Numbers of unique origin country(ies).
- **Destination country:** Numbers of unique destination country(ies).
- **AVG monthly weight:** Arithmetic mean of weight.

$$\text{Average monthly weight} = \frac{\sum_{i=1}^n \sum_{j=1}^m \text{weight}_{ij}}{n}$$

, where i is the number of months, j is the weight break.

- **AVG gross/net yield:**

$$\text{Average gross yield} = \frac{\sum_{i=1}^n \sum_{j=1}^m \text{charges}_{ij} + \text{surcharges}_{ij}}{\sum_{i=1}^n \sum_{j=1}^m \text{weight}_{ij}}$$

$$\text{Average net yield} = \frac{\sum_{i=1}^n \sum_{j=1}^m \text{charges}_{ij}}{\sum_{i=1}^n \sum_{j=1}^m \text{weight}_{ij}}$$

, *charges* are fees directly related to the weight of the shipment and *surcharges* contain all the other fees (fuel surcharges, security surcharges, taxes, etc.)

3.1.2 Top airports/air routes

When the data loading procedure is complete, you can go to the following tabs to see some visualization output. After clicking on the “Yield Analysis” tab, **CDM** will automatically proceed and visualize the loaded dataset.

3.1.2.1 Instruction

The first part of tab contains visualization of the **top n airports and air routes**. Figure 6 shows in the map the top 20 origin and destination airports. The surface of the circle represents the weight of shipment going through this airport. The darkness of the color represents the yield of the airport. The charts are

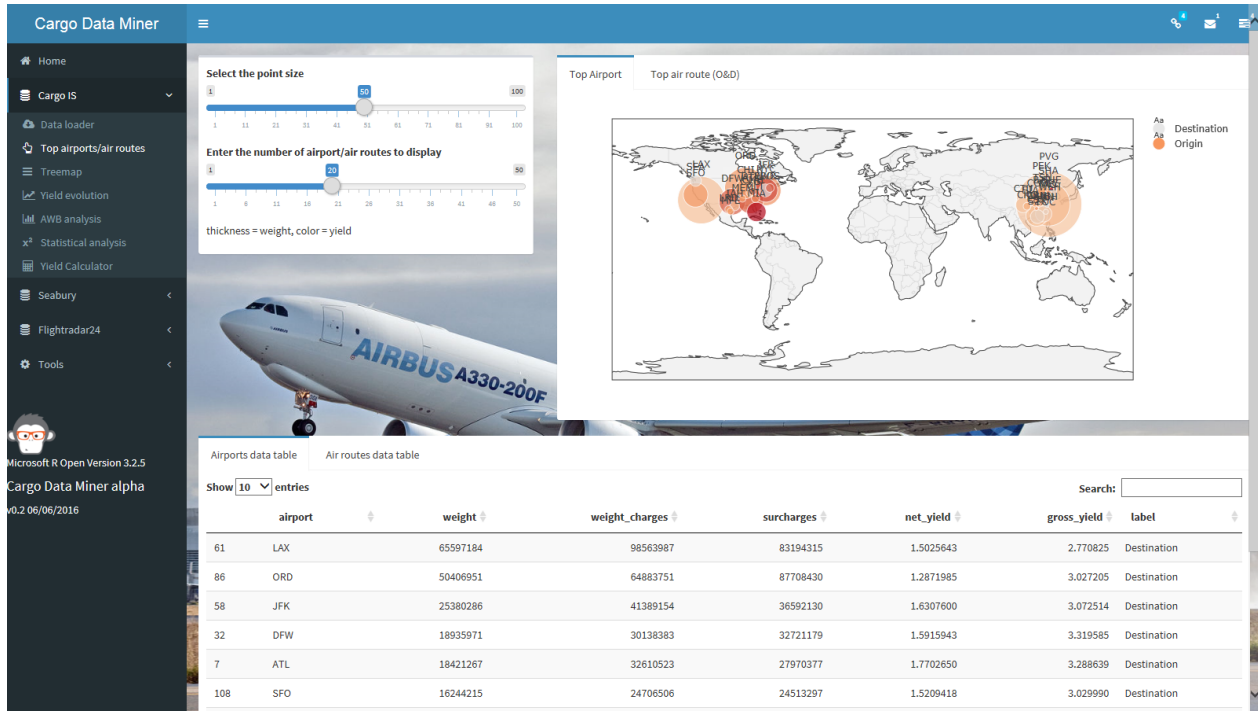


Figure 6: Yield Analysis tab, top 20 airports of CN-US

interactive, you can zoom in/out with the mouse scroll wheel. You can also click on the legend to display/hide the origin/destination airports.

The value of n could be modified by dragging the slider bar. You could also modify the size of the circle if sometimes the default size is too big/small.

The **Top air routes (O&D)** tab of the visualization chart shows the weight and yield of the most important air routes. Origin and destination airports are linked by blue curves with different thickness and darkness. The **thicker** the curve, the **more** the total weight and the **darker** the blue, the **higher** the yield.

The last two tabs (Airports data table and Air routes data table) show the two data tables that **CDM** uses to create the map. It could be sorted by variables and it could be directly copy-pasted into an Excel sheet.

3.1.2.2 Explanation

An R package `Plot_ly` was used to create the visualization map. From `/Scripts/Plot.airport.html.R`, an user defined function `Plot.airport.html()` is called.

```
#' [Plot.airport.html] could be used to visualize the top n airports/air routes
#' When the parameter table = TRUE, it will output the datatable that it will
#' use for the data visualization.
#'
#' @param dataset: data frame with five columns
#' c('ORIGIN_AIRPORT_CODE', 'DESTINATION_AIRPORT_CODE', 'WEIGHT_CURRENT_YEAR',
#' 'WEIGHT_CHARGES_CURR_YEAR_USD', 'OTHER_CHARGES_CURR_YEAR_USD')
#'
#' @param projection: Mercator or orthographic
#'
#' @param top: numbers of airport/air routes to be displayed
#'
```

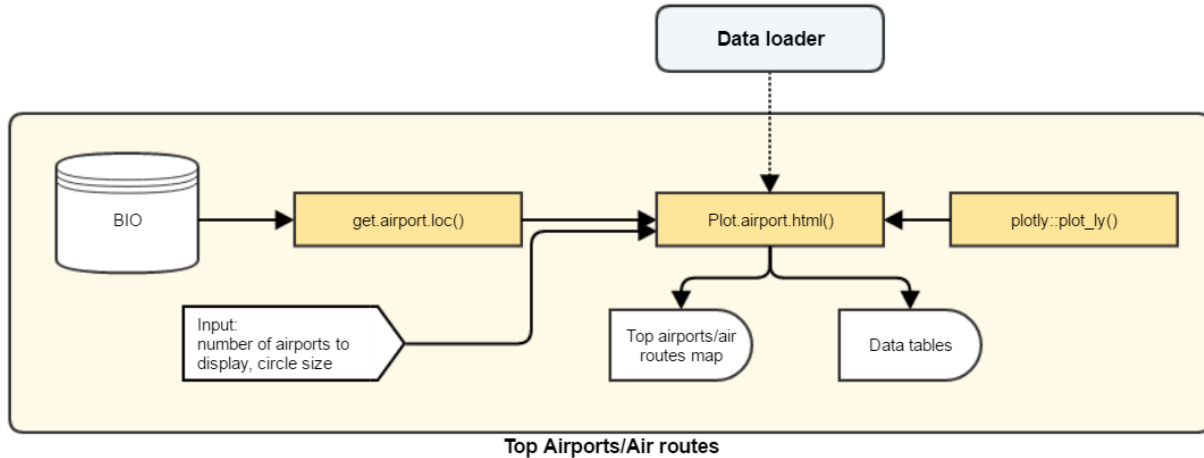


Figure 7: Flowchart of top airports/air routes tab

```
## @param plot.airport: if TRUE, plot airport, if FALSE, plot air routes
##
## @param pt.size: int, point size
##
## @param table: if TRUE, output datatable, otherwise, output map

Plot.airport.html(dataset, projection = 'Mercator', top = 20, plot.airport = TRUE,
                  pt.size = 800000, table = FALSE)
```

`Plot.airport.html()` will call `plotly::plot_ly()`. `plotly::plot_ly()` provides a simple way to add scatters/lines/curves to a map. It will only need the coordinates of the O&D airports, which we could easily get from BIO database by doing

```
source('/Scripts/get.airport.loc.R')
get.airport.loc()
```

Figure 7 shows how the function `plot.airport.html()` works.

3.1.3 Treemap

3.1.3.1 Instruction

The tab **Treemap** visualizes the relationship between different **nodes** (airports, countries and regions). Each node is displayed as a rectangle, sized by the total airfreight weight of this node and colored by the average gross yield of the node.

You can click on the rectangle to see its child nodes² (if exist), and right-click to see its parent node. A scale is shown on the top right to indicate the yield level. The tab **Table** will render the used data table for the visualization. The table contains four columns, *Name*, *Parent*, *Weight*, *GrossYield*.

Currently there is no fitable solution to generate a static downloadable image. So you may take a screen shot if you want to add the chart to your slides.

²For example, **country** is the child node of **region**, and it is the parent node of **airport**.

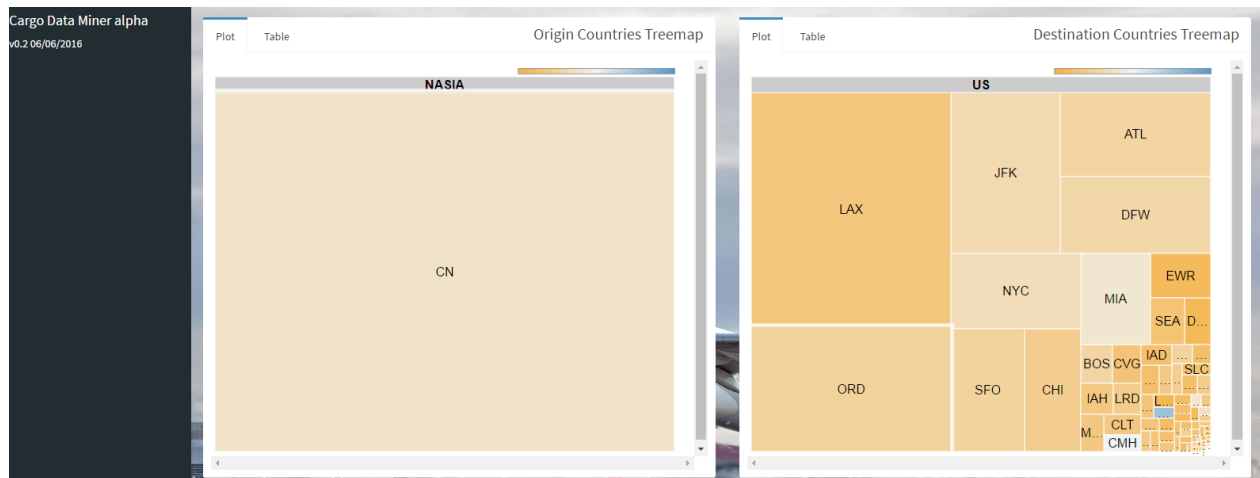


Figure 8: Treemap of airports for CN and US

3.1.3.2 Explanation

To create an interactive treemap, I created an user-defined function `treemap_cargois_gvis()`, which will call `googleVis::gvisTreeMap()`. The package `googleVis` is the R interface to **Google Charts API**. It generates an HTML code which could be integrated into R Shiny application. It could be further customized by using Javascripts.

```
# Example:
treemap_cargois_gvis(data, ORG = FALSE, PLOT = TRUE)

#' data is a data.frame object generated by Cargois.SQL.query()
#' If ORG = TRUE, then create treemap for the origin airports,
#' else for the the destination airports
#' If PLOT = FALSE, then the output will be a data.frame object
#' (instead of a plot)
```

The function `treemap_cargois_gvis()` will convert the queried Cargo IS data table into a four-column dataset (*Name*, *Parent*, *Weight*, *GrossYield*), which could be used by `googleVis::gvisTreeMap()`.

```
# Example of gvisTreeMap
gvisTreeMap(data = datatable, idvar = 'Name', parentvar = 'Parent', sizevar = 'Weight',
            colorvar = 'GrossYield',
            options=list(fontSize=16,
                        minColor='#F5B041',
                        midColor='#F0F3F4',
                        maxColor='#5499C7',
                        headerHeight=20,
                        fontColor='black',
                        showScale=TRUE)
            # options, see
            # https://developers.google.com/chart/interactive/docs/
)
```

3.1.4 Yield Evolution

Yield is the most important information provided by Cargo IS database.

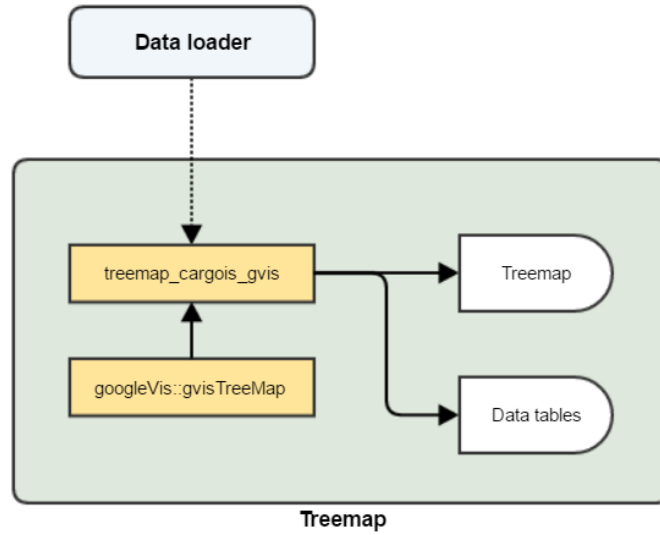


Figure 9: Flowchart of Treemap tab

3.1.4.1 Instruction

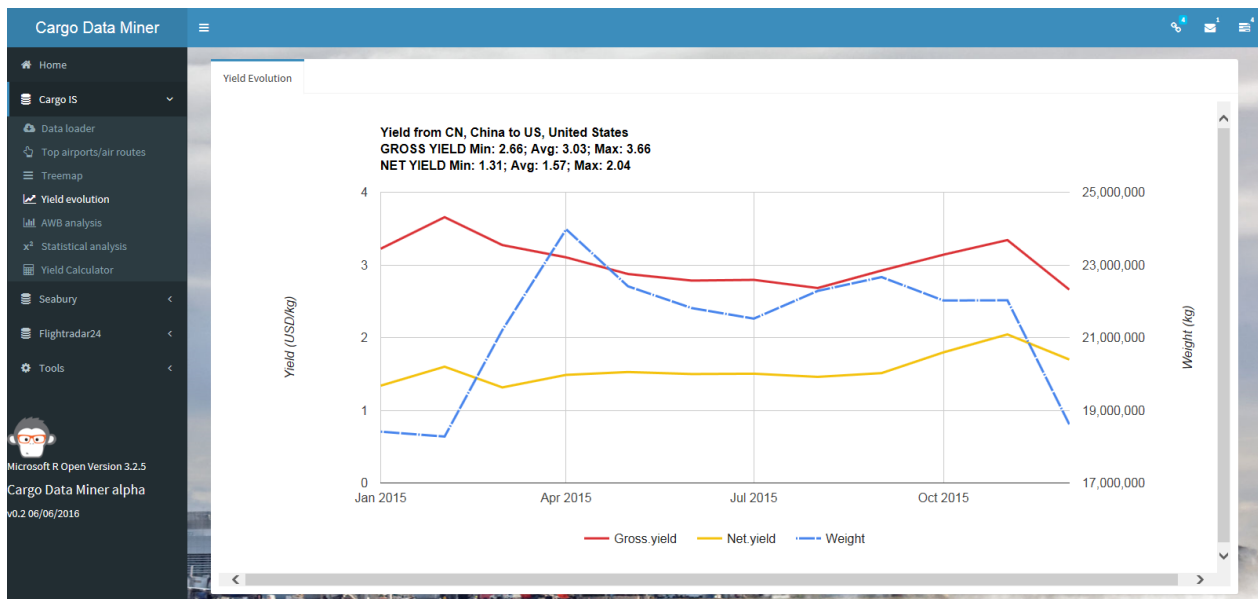


Figure 10: Airfreight weight and yield evolution for CN and US

The next tab **Yield evolution** generates an airfreight evolution plot. **CDM** calculates and render a line chart showing the evolution of gross yield, net yield and airfreight weight during the selected period of time. The minimum, maximum and average value of the output are shown in the plot's subtitle.

The x-axis represents the available months during the selected period. If the box "Plot evolution since 2010" in the data loaded page is checked, all the available data will be visualized (but the computing will also take a longer time).

You could click on the curve to see detailed information of the choosen data point.

A data table appears on the bottom of the page. It contains all the data of the line chart. You could click on

the download button to save it to the local.

3.1.4.2 Explanation

I would like to integrate the three curves (net yield, gross yield and weight) into one single chart. Thus two different y-axis are needed. The well known package `ggplot2` does not support two different y-axis in one chart so I will still use `googleVis` to generate the line chart.

```
# Example of gvisLineChart
# with gvisLineChart, we could easily create a line chart with 2 y-axis

gvisLineChart(data, xvar = "xaxis", yvar = c('yaxis_1', 'yaxis_2', 'yaxis_3'),
              options = list(series="[
                                {targetAxisIndex: 0}, # set the target y-axis
                                {targetAxisIndex: 0},
                                {targetAxisIndex: 1}
                                ]"
              ))
```

3.1.5 Air Waybill Analysis

An **air waybill** is a receipt issued by an international airline for goods and an evidence of the contract of carriage. It is the most important document issued by the carrier (directly or through an authorised agent). Cargo IS contains the records the weight data and it regroups the air waybills into six categories, which are called **weight breaks**, regarding to its weight.

They are:

- 0 to 50 kg
- 50 to 100 kg
- 100 to 300 kg
- 300 to 500 kg
- 500 to 1000 kg
- larger than 1000 kg

The proportion of the AWB of different categories in a market could be an indicator of the economic situation. For example, an increasing number of heavy shipments often means the cargo consolidation becomes more efficient, and the economic situation is going better.

3.1.5.1 Instruction

The first row of this tab contains seven figures:

- *Shipment frequency evolution* is a line chart which shows how the numbers of shipments change during the selected period. By default, there will be six curves representing respectively the different weight break.
- *Shipment weight distribution* is a bar chart which shows the distribution of the AWB of different weight break in the selected period.
- *Weight vs. distance* is a stacked bar chart showing the sum of shipment weight by different intervals of distance. The colors represent the different weight breaks.
- *Frequency vs. distance*: like the previous figure, it shows the numbers of shipments by different intervals of distance.

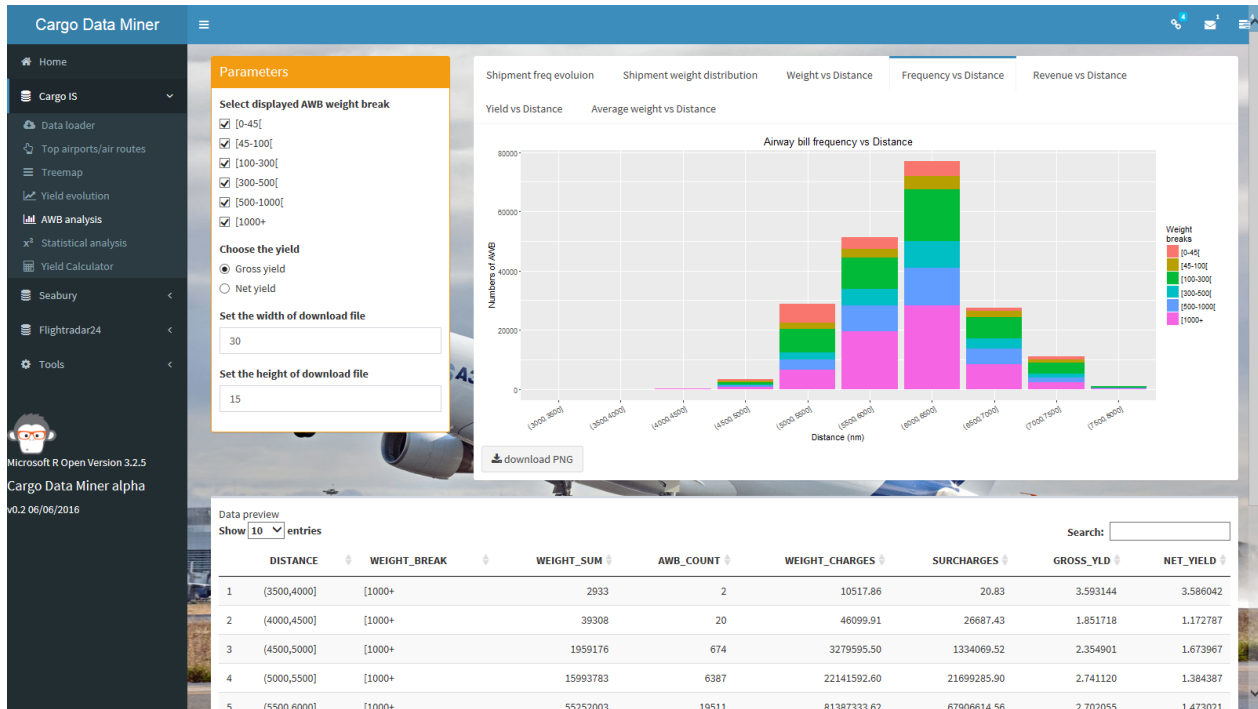


Figure 11: Air waybill analysis

- *Revenu vs. distance* shows the revenues (charges) of airfreight by different intervals of distance.
- *Yield vs. distance* is a line chart showing the gross/net yield of by weight breaks and by distance.
- *Average weight vs. distance* takes into account all the weight breaks and compute the weighted arithmetic mean of weight by distance.

For the charts 3, 4, 5 and 6, if there is only one interval of distance, a pie chart will be created instead of a bar chart. All the charts can be downloaded by clicking “download PNG” button.

On the left of the charts, some parameters can be changed to modify the charts. You can display/hide one or multiple weight break(s) by checking/unchecking the boxes. The second part “gross yield/net yield” corresponds to the yield type in the chart *Yield vs. distance*. The last two text input areas control the width and height of the downloaded image.

On the bottom of the page, the data table used for creating the charts is displayed. It could also be downloaded by clicking the download button.

3.1.5.2 Explanation

Before creating the charts, data manipulation should be done in order to have the right data to plot. `data.table` is a very powerful package for big data computing.

Example of data.table

```
system.time(
  output <- dt[,.(weight = sum(WEIGHT_CURRENT_YEAR),
    freq = sum(AWB_COUNT_CURRENT_YEAR),
    wchg = sum(WEIGHT_CHARGES_CURR_YEAR_USD),
    surchg = sum(OTHER_CHARGES_CURR_YEAR_USD),
    gy = (sum(WEIGHT_CHARGES_CURR_YEAR_USD)+
```



```

        sum(OTHER_CHARGES_CURR_YEAR_USD))/sum(WEIGHT_CURRENT_YEAR),
ny = (sum(WEIGHT_CHARGES_CURR_YEAR_USD))/sum(WEIGHT_CURRENT_YEAR)
), by = .(dist = lebal, wb = WEIGHT_BREAK)
]
)
# dt is a 1 750 000*15 data.table,
# user system elapsed
# 0.14    0.00    0.14

```

To create these charts, the package `ggplot2` is extensively used.

3.1.6 Yield Calculator

The last subtab of Cargo IS is a marketing tool called yield calculator, which offers an automatic computing for air route cargo yield. It will read air routes from a `csv` file (uploaded by user), query *Cargo IS* database and display a data table with weight and yields of every air route in the uploaded list, as well as a air route map visualizing all the air routes by their cargo weight and cargo yield.

3.1.6.1 Instruction

The idea of yield calculator is to get rid of manual cargo yield computing. To use yield calculator, you will need to create a `csv` file containing two columns: “From” and “To”, which correspond to the origin and destination airport. Once the `csv` is ready, you could upload it to **CDM** by clicking **choose file** button. Then, the year parameter should be selected. The others parameters could be modified if according to the format of your `csv` file. When everything is done, click on *Start*.

A data table will be displayed after the computing. The gross yield, net yield, weight, gross charges and net charges will be calculated for each air route. On the bottom, an air routes map will show the input air routes by the cargo weight and cargo yield. The darker the curve, the higher the yield. The thicker the curve, the higher the cargo weight. Click on “Download” to download the data table. It will be a `csv` file as well.

There may be rows only containing value **zero** in the output table. It is because of the lack of data in Cargo IS database. We do not have any information about the given air route.

On the left of the map, there are some display parameters. You can select the appropriate map projection, the comparing criteria and some graphical parameters to customize the map.

3.1.6.2 Explanation

Figure 13 show the data manipulation procedure. Two input are used. The first one is the uploaded `csv` file and the second one is the Cargo IS data of the selected year (queried by `CargoIS.SQL.query()`). The function `grs_net_yield2()` calls the package `data.table` and computes cargo yield for every air route of this data table. The yield table will be left joined to the uploaded air route list. After merging, the data table will be displayed on the top right of the page and the function `Plot.air.route.html()` will be call to further generate a map.

3.2 Seabury Database

The second part of this document consists analysis and visualizations of **Seabury Trade** database. Seabury Trade contains annual country level cargo weight and value information. It has not only the airfreight weight and value, but also for the other surface transportation. It provide an in-depth level of commodity type which could allow us to get some knowledge about the cargo type of the market.

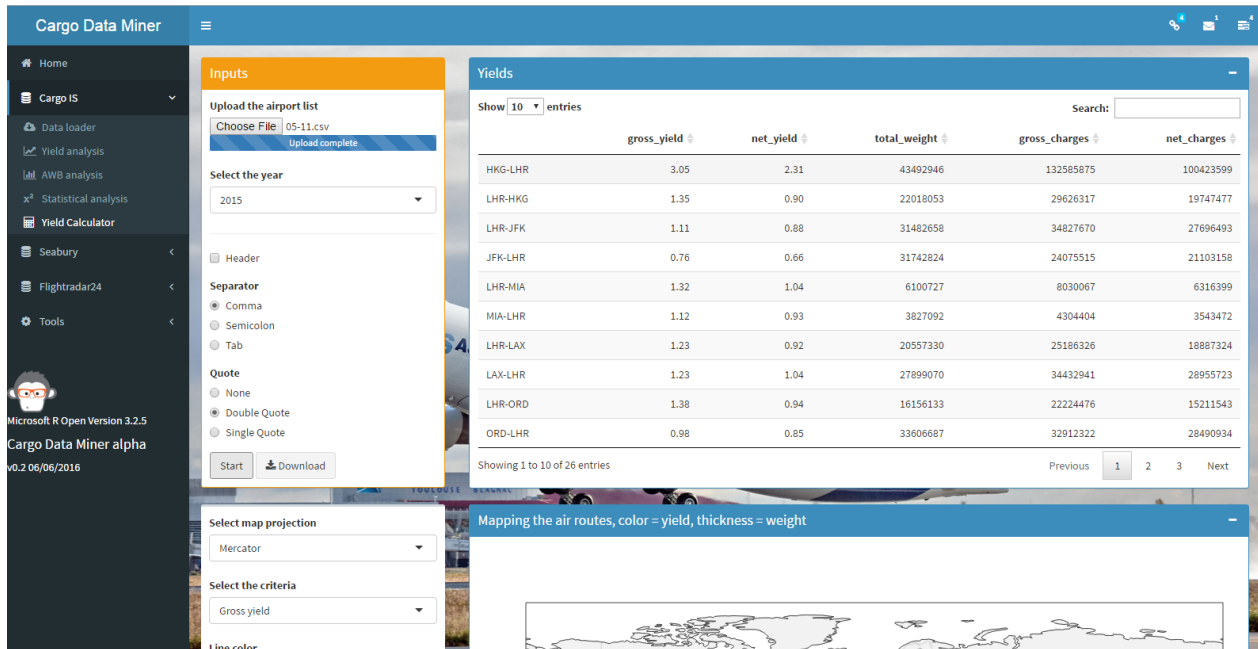


Figure 12: Yield calculator interface

Beside Seabury trade, there two other databases which could be useful for market knowledge. They are Seabury Express and Seabury. They could be very useful for the further development of the application.

Seabury databases are stored in the same data server as Cargo IS.

3.2.1 Data loader

3.2.1.1 Instruction

As for Cargo IS query, the first step here also consists of data loading. You must select firstly the data granularity to be queried. Then, the year parameter should be set as well. Seabury Trade database contains annual data from 2000. Then the drop menus of *Origin* and *Destination* will change automatically according to your selection of data granularity. When all the input parameters are set, click on the **Run** button to start to query the database.

3.2.1.2 Explanation

The start button triggers the user-defined function `Seabury.SQL.OD.query()`. It works just like `CargoIS.SQL.query()`: it will generate a SQL query that depends on the user input. Then it send the generated query to our SQL server and fetch the returned data.

```
# Example
#' ODBC: a Oracle SQL connector,
#' level: granularity of queried data
#' ORG: Origin
#' DST: destination
#' Year: interger of NA, if NA, query all the data, otherwise query the selected year
#' country.name: if TRUE, return the full country name
Seabury.SQL.OD.query(ODBC, level, ORG, DST, year = NA, country.name = FALSE)
```

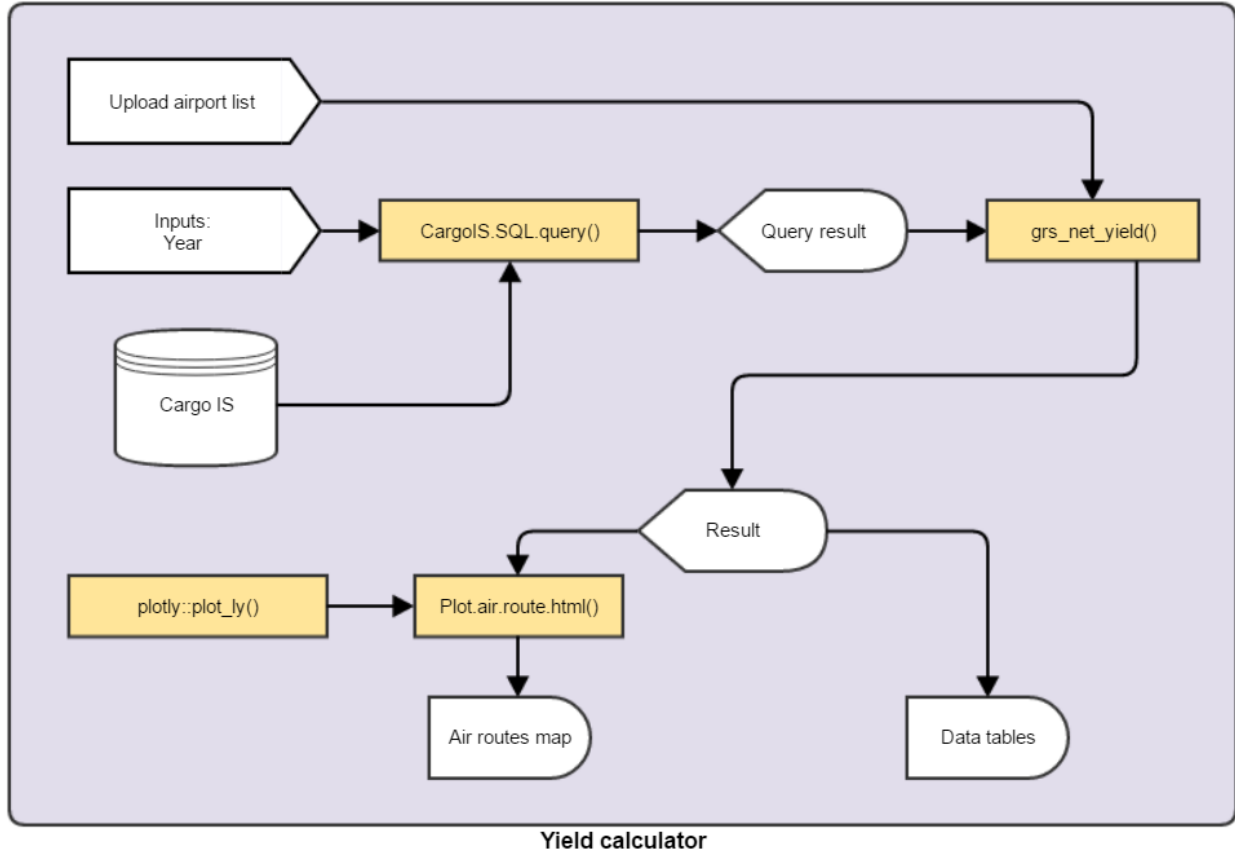


Figure 13: Flowchart of yield calculator

3.2.2 Commodity treemap

Seabury Trade database contains a detailed commodity categorization system. It has eight main categories and more than 2000 sub-categories.

3.2.2.1 Instruction

Seabury Trade has more than 70 000 000 rows so the data loading process may take longer time than Cargo IS. When data is loaded, **CDM** will generate a treemap to present the commodity types, their weight (in kg) and their value (in USD). The size of the square corresponds to the cargo weight for this type of goods. Its color represent the value. The data table could be view in the *Table* tab, and you could download it by clicking on the button *Download*. The structure of the data table is the same as the airport treemap in Cargo IS.

You can click on the square to look further into the sub-categories of the selected commodity type. There are four different levels of categories, G1, G2, G3 and G4. The bigger the number, the higher the complexity. G1 level contains eight different categories but there are around 2000 in the G4 level.

You could change to look at the ocean freight information by simply selecting the freight type from the dropdown menu. The data table is also downloadable by clicking **Download** button.

3.2.2.2 Explanation

To draw the treemap, a user-defined function `treemap_dt_sea_gvis()` is firstly called to generate a data table that fits the requests of `googleVis::gvisTreeMap()`. Then the treemap is generated with a user-defined function `treemap_sea_gvis()`, which will call `googleVis::gvisTreeMap()`.

```
# Function to generate data table for treemap of SEABURY GXCODE
#' data: an output datatable of Seabury.SQL.OD.query()
#' if AIR = TRUE then output airfreight value
#' otherwise, output surface freight value
treemap_dt_sea_gvis(data, AIR = TRUE)

#' Function to draw the treemap of SEABURY GXCODE
#' data: Output of treemap_dt_sea_gvis()
#' ORG, DST, YEAR: parameter of the title of the treemap
treemap_sea_gvis(data, ORG, DST, YEAR)
```

3.2.3 Evolution plot

3.2.3.1 Instruction

The second major functionality of Seabury Trade tab is the cargo weight evolution plot. By selecting the name of commodity type, you can thus create a line chart that represents the evolution of airfreight weight since 2000. If the box “Plot all level” is checked, then all the data base will be loaded and visualized (This could take a very long time).

There are four sub-tabs (G1LEVEL, G2LEVEL, G3LEVEL and G4LEVEL) representing the four different catogiries. The evolution of the sub-categories (if exist) of the selected type will be plotted following sub-tab. You can change the y-axis from cargo weight to cargo value.

The data table will be displayed in the last sub-tab.

Attention: If you selected, for example, a G3 level category, then please go the the third sub-tab to get the correct line chart. The charts in the superior levels will be the same as the third tab, but the commodity name will be wrong. To know the level of category that you selected, you can count the number of “-” before the name of category.

3.2.3.2 Explanation

3.3 FlightRadar 24 Database

FlighRadar 24 is a recently purchased database at Airbus. It provides flight tracks, origins and destinations, flight numbers, registration numbers, flight positions (coordinates), altitude, heading and speed for 2014 and 2015. The data of FR24 mainly comes from the third party ADS-B (Automatic dependent surveillance-broadcast) receivers all over the world. The receivers collect data from any aircraft in their local aera that are equipped with an ADS-B transponder and upload to the internet in the real time.

FR24 could be very valuable for Airbus freighter because it may fill the gap of the lack of express cargo airlines' data. With FR24, we could extract the schedule of these airlines, which we did not have any information before.

But, comparing with other databases, it has some particularities.

- We have two stock location for this database. The first one is the **MySQL** server inside airbus (which is used by **CDM**). The second one is the cloud-based platform **Foundry**.
- The data size is way larger than the other databases. Because it records the real-time position of aircraft, so we have about 1-2 Gb of data per day.

- The data quality of FR24 is a major challenge for analysis. For example in the column “Aircraft Type”, we have several different codes representing the same aircraft (*e.g.* both B77F,B77L,77L and 77F represent Boeing 777 freighter)

3.3.1 Data loader

3.3.1.1 Instruction

3.3.1.2 Explanation

3.3.2 Airline network

3.4 Other features

3.4.1 Connection status check

The first link icon on the top right shows the connection status of **CDM** to the databases. If all the databases are successfully connected, the color around the number will be blue. Otherwise the background color will turn red.

You can click on the icon to see the detailed connection information.

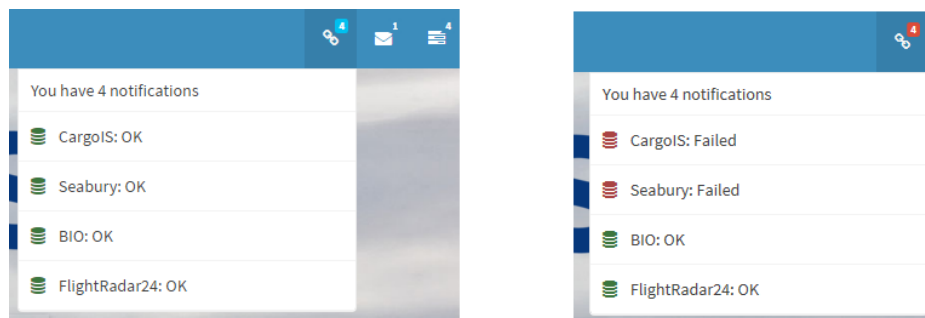


Figure 14: Connection succeed/failed

3.4.1.1 Explanation

Each time when a new session of **CDM** starts, it will try to make a connection to these databases and check the class of the connector by using `try()`. It will return “try error” in the case that the connection to databases could not be established.

```
CargoDB <- try(dbConnect(drv, username = username_cargois,
                        password = password_cargois,
                        dbname = connect_string_cargois))
```

If the class of all the connectors are not 'try-error', the value OK will be given to the variables `statu_cargois`, `statu_bio` and `statu_fr24`, which will determine the color and the text output of the header's notification.

IMPORTANT: Functions related to a database will be loaded only if the database could be reached.

3.4.2 Fast selectize input

4 Future Development

5 Appendix: Troubleshooting

5.1 Install Oracle Database instance client

Oracle Database Instant Client (11.2.0.3) EN_W764 could be found in **PC Service**, or it could be downloaded from <http://www.oracle.com/technetwork/topics/winx64soft-089540.html> (you may need to create an account to be able to download it). After downloading the client, extract it to a folder that you could find again.

If **CDM** could not be executed due to errors of Oracle instant client, you can check if the environment variables of Windows is set correctly by doing the following procedure.

- Open the properties window of the computer.
- Click on “**Advanced system settings**”
- Select the tab “**Advanced**” of the appearing dialog box.
- Click “**Environment Variables**” button on the bottom.
- Find the variable “**Path**”, make sure that the path of your Oracle instance client is added. If not, add it.

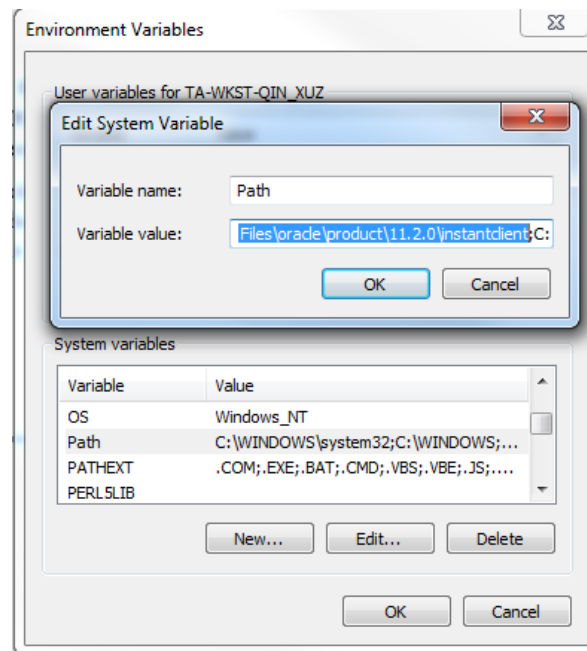


Figure 15: Check the "Path" system variable

If the problem still exists, check the **ROracle** package installation instruction on <http://cran.us.r-project.org/web/packages/ROracle/INSTALL>

5.2 Download and install an R package manually

Some of the R package could not be installed directly from **CRAN**, for example the package **ROracle**. In this situation, the manual installation will be needed.

Here I will show the installation process for **ROracle**.

- Firstly, download **ROracle_1.2-1.zip** from *oracle.com* (<http://www.oracle.com/technetwork/database/database-technologies/r/roracle/downloads/index.html>)
- Secondly, place the .zip file into `~/CDM/Rpackages` and make sure that *Oracle Instant Client* is correctly installed.
- Then, instead of running

```
install.packages('ROracle')
```

run

```
install.packages('Rpackages/ROracle_1.2-1.zip', repos = NULL, type="source")
```

After the installation, the package could be called by

```
library(ROracle)
```